Distributed Particle Simulation Using Heterogeneous Supercomputers

Jeffrey C. Becker[1] and Leonardo Dagum[2]

Report RND-91-012, November 1991

NASA
National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

ARC 275 (Rev Feb 81)

Distributed Particle Simulation Using Heterogeneous Supercomputers

Jeffrey C. Becker[1] and Leonardo Dagum[2]

Report RND-91-012, November 1991

NAS Systems Division
NASA Ames Research Center
Mail Stop 258-5
Moffett Field, CA 94035-1000

---

[1] Division of Computer Science, Univ. of California, Davis, CA 95616

[2] Computer Sciences Corporation, NASA Contract NAS 2-12961, Moffett Field, CA 94035

Distributed Particle Simulation Using Heterogeneous Supercomputers

Jeffrey C. Becker*
Division of Computer Science
University of California
Davis, CA 95616

Leonardo Dagum†
NASA Ames Research Center
M/S T045-1
Moffett Field, CA 94035

November 4, 1991

Abstract

We describe the implementation and performance of a three dimensional particle simulation distributed between a Thinking Machines CM-2 and a Cray Research Y-MP. These are connected by a combination of two high-speed networks; a High Performance Parallel Interface (HIPPI) attached to the CM-2, and an optical network (UltraNet). This is the first application to use this configuration at NASA Ames. We describe our experience implementing and using the application and report the results of several timing measurements. We show that the distribution of applications across disparate supercomputing platforms is feasible and has reasonable performance. In addition, several practical aspects of the computing environment are discussed.

*Supported by a DARPA/NASA Research Assistantship in Parallel Processing administered by the Institute for Advanced Computer Studies, University of Maryland

†The author is an employee of Computer Sciences Corporation. This work is supported through NASA contract NAS-2-12961.

1 Introduction

Current supercomputer speeds and memory capacities limit the size of applications that can be run on them. Heterogeneous Distributed Supercomputing (HDS) promises to alleviate this shortfall by combining the resources of disparate supercomputing platforms through high-speed networks. This natural extension of supercomputers has only recently become feasible due to advances in network technology. The Numerical Aerodynamic Simulation (NAS) facility at NASA Ames includes several supercomputing platforms connected by high-speed networks. Thus we were motivated to try a series of experiments using HDS.

In addition to increasing the amount of processor and memory resources available to an application, HDS provides several other advantages over stand-alone supercomputers. Most importantly, it allows an efficient distributed implementation of an application comprised of several heterogeneous components, each with different computing requirements. An example is a simulation of coupled systems in an entire aerospace vehicle. HDS allows each part of a distributed application to be run on an architecture for which it is best suited. For example, certain codes vectorize well while others are more appropriate for a Multiple Instruction Stream Multiple Data Stream (MIMD) parallel supercomputer such as the Intel IPSC/860. An HDS system also provides parallelism by running different components simultaneously on separate platforms.

Despite its potential, HDS also has a number of possible pitfalls. In particular, the networks must be capable of adequate transfer rates in order to sustain high computing speeds on the supercomputers. It is wasteful of processor time to be waiting for network I/O. In addition, the data format of one machine is not likely to be identical to that of another. Thus, the amount of time spent transforming or translating data must be measured. For HDS to be viable, this should not be a significant fraction, since it figures into the overall effective bandwidth. If transfers between two machines must cross several networks, then there might be potential mismatches such as packet size and transfer rate differences that could affect the overall transfer rate. There should exist high level programming abstractions that facilitate use of HDS. Finally, HDS applications should be able to run effectively given other user loads on the system.

We have implemented a three dimensional particle simulation application

on an HDS system comprised of a 32K processor Thinking Machines (TMC) CM-2 and a Cray Research Y-MP connected by a combination of a High Performance Parallel Interface (HIPPI) and a high-speed optical network (UltraNet), as well as a direct Ethernet link. We chose this application for our initial experiments because one part of its original implementation on the CM-2 could be run more efficiently on the Y-MP. Our main result is that HDS is feasible for this application although there are several areas in which improvements can be made. Section 2 describes some related work at the Pittsburgh Supercomputing Center (PSC). Section 3 describes the particle simulation application and its implementation and use. Experiments and results are described in Section 4. Finally, Section 5 contains conclusions and suggestions for future research.

2 Previous Work at the Pittsburgh Supercomputing Center

Schneider [Schneider91] describes a distributed chemical engineering application at PSC by professor George McCrae and graduate student Robert Clay of Carnegie-Mellon University. Their hardware platform is almost identical to the one we use, namely, a 32K processor Thinking Machines CM-2 and a Cray Research Y-MP connected by both high-speed and Ethernet networks. However, their high-speed network connection is a direct HIPPI link, while our system is connected by a combination of HIPPI and UltraNet.

Their system also shares several features with our particle simulation. Both implementations use two network connections. The high-speed HIPPI (-UltraNet) link is used for large data transfers, while the slower Ethernet link is used to transfer control information. In addition, both applications are task serial. The algorithms involve some initial computation on the CM-2, followed by a transfer of data to the Y-MP. The latter then performs its part of the computation while the CM-2 waits to receive data so that it may complete the computation. A task serial application does not benefit from the parallelism to be gained by running code on two supercomputers simultaneously. However, it highlights the effect of network bandwidth, since it is not possible to hide the network latency by techniques such as pipelining.

The PSC group encountered a number of the pitfalls mentioned earlier.

They found that the CM I/O bus limits the attainable network bandwidth since it is slower than HIPPI. However, their setup only includes one such bus. The NAS system has two busses, and thus, twice the available bandwidth. In addition, a significant fraction of time was used to convert data between the CM-2's parallel format, and the Y-MP's serial format. Despite these problems, their distributed application gains a speedup of ten over elapsed times attainable on either supercomputer individually. Thus their experiences indicate that HDS is potentially viable for our particle simulation application as well.

3 Three Dimensional Particle Simulation

3.1 Original CM-2 Implementation

Direct particle simulation methods are important techniques for analyzing hypersonic flows. Our implementation is a port of a three dimensional particle simulation developed in C/Paris for the CM-2 by Dagum [Dagum91]. Each particle is mapped to a single virtual processor. Since the number of virtual processors represented by each physical processor is the same, this mapping provides a good load balance. Each time step of the simulation involves the following steps. Details may be found in [Dagum91].

1. Collisionless motion of particles.
2. Enforcement of boundary conditions.
3. Sorting of particles into cells.
4. Pairing of collision partners.
5. Collision of selected collision partners.
6. Sampling for macroscopic flow quantities.

The original implementation for the CM-2 is highly optimized for data parallel operation, and communication is minimized as much as possible. Despite this, the most expensive step is the sorting of particles into cells. It consumes 38% of the computational time for a problem of size 32 million particles. Thus we were motivated to move the sorting step to the Y-MP, where a fast sequential bucket sort could be employed.

3.2 Distributed Implementation

The distributed particle simulation involves additional code for sorting the particles on the Y-MP, and for network communication. The distribution of work is shown in Figure 1. New steps are italicized.

CM-2	Y-MP
Move particles	
Enforce boundary conditions	
Send cell index keys	
	Receive cell index keys
	Sort keys
	Send particle ranks
Receive particle ranks	
Pair collision partners	
Collision of partners	
Sampling	

Figure 1: Distributed Particle Simulation Outline

We used the system architecture shown in Figure 2 to distribute the application. The CM-2 is connected to the CM-HIPPI through its I/O busses. The HIPPI connection connects the CM-HIPPI to the UltraNet Hub which then connects to the Y-MP over UltraNet. There is also a direct Ethernet link between the CM-2 front end and the Y-MP. The peak bandwidths of the networks are given in Figure 2. The unlabeled network connection is the bus used by the front end to broadcast instructions to the CM-2 processors. It does not affect the transfer rate between the CM-2 and the Y-MP.

We removed the sorting step from the original CM-2 implementation and coded the new steps in C on the Y-MP and C/Paris on the CM-2. Although the FORTRAN compilers available on the Y-MP generally produce more highly optimized code than the C compilers, C is the only language on both the Y-MP and CM-2 that allows calls to the UNIX socket library used for network communications. Furthermore it was convenient to use C/Paris since the CM-2 code was already written in this language. Specifically, the new

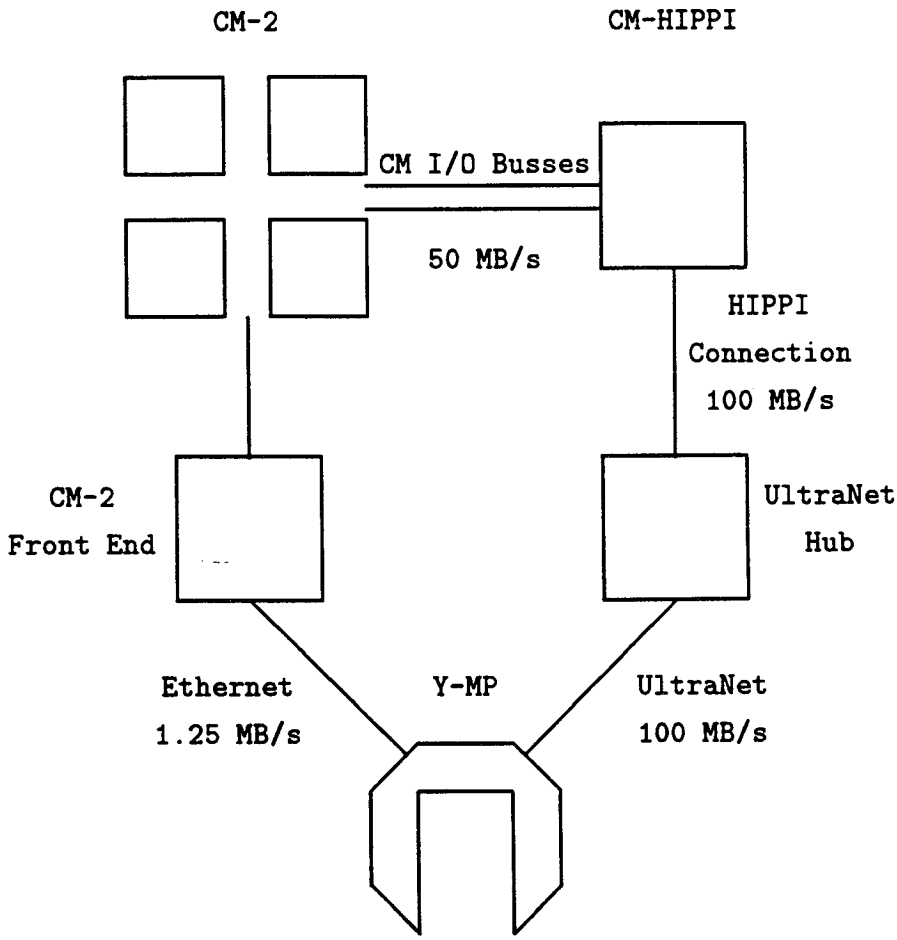

Figure 2: System Architecture

code includes a bucket sort procedure on the Y-MP and the familiar set of calls to the socket library.

As mentioned earlier, we use two sockets. One uses the HIPPI-UltraNet connection for high volume data transfers. The other uses Ethernet to transfer control information. In both cases, the Y-MP is the server and the CM-2 is the client. The control data is interactive input read and then sent from the front end of the CM-2. For example, this information is used to instruct the Y-MP code to perform a number of timesteps or to double the number of particles.

It is necessary to transpose the data from CM-2 parallel format to a serial format readable by the Y-MP. Calls to the CM-2 transpose procedure are required before sending data to and after receiving data from the Y-MP. In addition, the Y-MP's byte ordering is reversed from the serial order required by the CM-2. Thus, data are transmitted in the CM-2 serial order and (un)packed appropriately by the Y-MP. The detailed outline is shown in Figure 3 with new steps italicized. Note that only one set of socket calls is shown.

3.3 Running the Application

Running the application requires two windows, one for the CM-2 code and one for the Y-MP code. Since the Y-MP is server, its code must be started first. If the Y-MP is heavily loaded, the job often gets swapped out, causing socket requests from the CM-2 to timeout. It is difficult to synchronize on two sockets. For jobs bigger than four million particles, the Network Queuing System (NQS) is necessary to access the required memory and processor resources on the Y-MP. However, the job must still be run interactively. Thus a third window is used to run "top" on the Y-MP to see when the Y-MP job has started. It is only then that the CM-2 job can begin. This is somewhat clumsy. The application could be implemented to better fit the NAS environment. However, if this sort of job is to be run with any frequency in the future, the computing environment should be modified to give additional consideration to HDS applications.

CM-2	Y-MP
Initial Code	
	socket
	bind
	listen
	accept
CMFS_socket	
CMFS_connect	
Timestep Code	
Move particles	
Enforce boundary conditions	
Transpose cell index keys	
Send cell index keys	
	Receive cell index keys
	Unpack cell index keys
	Sort keys
	Pack particle ranks
	Send particle ranks
Receive particle ranks	
Transpose particle ranks	
Pair collision partners	
Collision of partners	
Sampling	

Figure 3: Detailed Distributed Particle Simulation Outline

3.4 Initial Problems

As our application is the first to use the HIPPI-UltraNet link between the CM-2 and the Y-MP, we uncovered several problems. At first, we could not connect to the CM-2 socket server daemon. This was a result of several hardware problems including a faulty CM I/O processor board and broken I/O bus wires. After replacing these faulty parts, we still could not make our network connection due to a faulty configuration file. This file is now fixed and we can complete our connection. However, the network still hangs periodically. In addition, we have been unable to run our application from all four sequencers. The TMC engineers believe this might be due to I/O bus problems. They are looking into these problems with the help of Ultra engineers.

4 Experiments

We ran several problem sizes ranging from 65,536 to 2,097,152 particles, doubling the number of particles from one run to the next. Each problem size was run twice; once from one and once from two sequencers on the CM-2. Each sequencer is attached to 8,192 physical processors. Doubling the number of sequencers allows the use of two I/O busses instead of just one. This allows a larger network bandwidth to be attained. Each run consisted of ten time steps. For a given problem size, each CM-2 virtual processor sends a four-byte cell index to the Y-MP every time step and receives a four-byte rank. Thus each time step is comprised of two transfers, each of size (in bytes) equal to the number of particles multiplied by four. All the graphs have time measured in seconds on the y axis and problem size measured in MParticles on the x axis. One MParticle is equal to 1,048,576 particles.

Figure 4 shows total times for the single sequencer runs of both the distributed and stand-alone CM-2 implementations. The times for the stand-alone CM-2 runs vary linearly with problem size. The distributed run times are approximately constant for small problem sizes, and then vary linearly for problem sizes larger than 0.5 MParticles. At small problem sizes, the distributed version must overcome a large network overhead to transfer a relatively small amount of data. This overhead dominates the transfer time and causes the execution time to be constant. Despite the fact that the dis-

tributed code takes longer than the code running on the CM-2 alone, it is well within a factor of two for problem sizes larger than half a million particles.

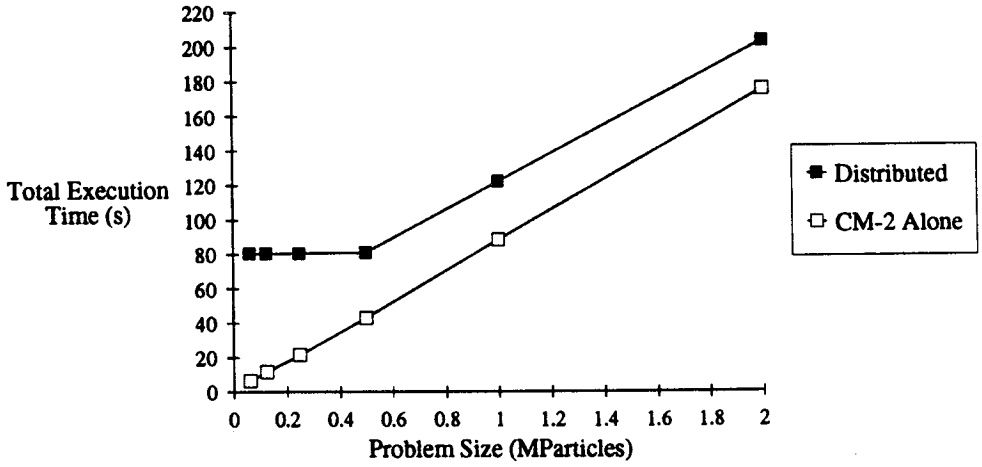

Figure 4: Total Execution Time, One Sequencer

The effect of doubling the sequencers, and hence the number of I/O busses is reflected in the total times shown in Figure 5. As in Figure 4, there is a large overhead at small problem sizes for the distributed code. This is not surprising since the small transfer sizes are unlikely to be affected by the increase in bandwidth due to the additional I/O bus. Comparing Figures 4 and 5, there is a greater disparity between the two implementations for the double sequencer runs than for the single sequencer runs. This is due to the fact that the CM-2 stand-alone performance is doubled by the addition of another sequencer, while the distributed code only has performance boosted in the CM-2 and network transfer components. In fact, only the largest problem size seems to have improved performance over the single sequencer version. For a given problem size, Y-MP performance is unaffected by the addition of another sequencer because the amount of data is the same as in the single sequencer case.

Given Figures 4 and 5, it is not possible to say whether the distributed implementation will ever surpass the performance of the stand-alone implementation. Further experimentation with larger problem sizes is needed.

The effective sorting times for the single sequencer runs are compared in Figure 6. For the distributed implementation, these times consist of the

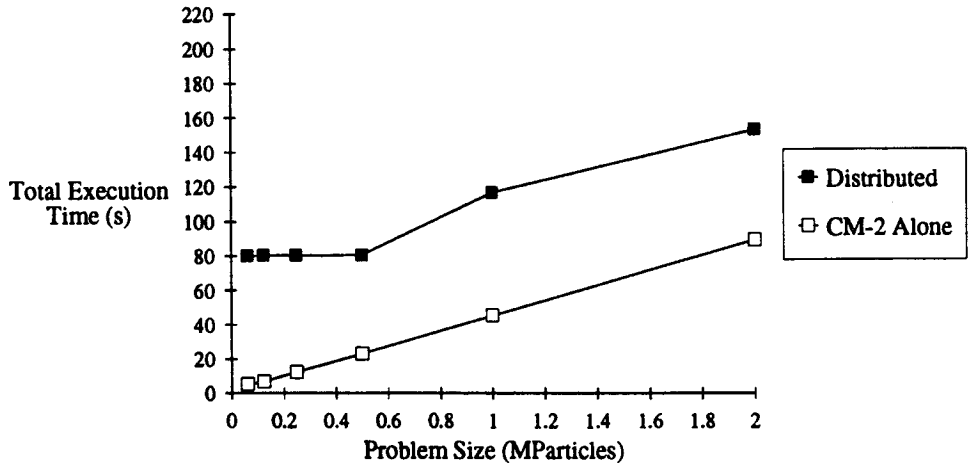

Figure 5: Total Execution Time, Two Sequencers

sorting times on the CM-2 and the Y-MP, as well as the data conversion and network transfer time. Since the sorting time on the CM-2 also includes the time to permute the particle data, this permute time must be subtracted from both the distributed and stand-alone CM-2 times to compute the effective sorting time. The stand-alone CM-2 times vary linearly with problem size. This is expected for the sorting algorithm employed [Dagum91].

It is interesting that for small problem sizes, the effective sorting time of the distributed implementation actually drops as the problem size increases. The bucket sort used on the Y-MP in the distributed code is a linear time algorithm, and is faster than the stand-alone CM-2 sort. However, the total of the Y-MP sorting time and network transfer time is approximately constant for small problem sizes despite the growing sort and permute times. This is due to the apparent drop in network transfer time with problem size, as shown in Figure 9.

It appears that for larger problem sizes, the distributed curve is parallel to the stand-alone CM-2 curve. This indicates that distributed sorting will never be as fast as the stand-alone CM-2 version. However, additional runs with larger problem sizes are required to confirm this.

Sorting times for the double sequencer runs are shown in Figure 7. The overall features of this graph are similar to that of the previous graph. Although the stand-alone CM-2 times are reduced by a factor of two over the

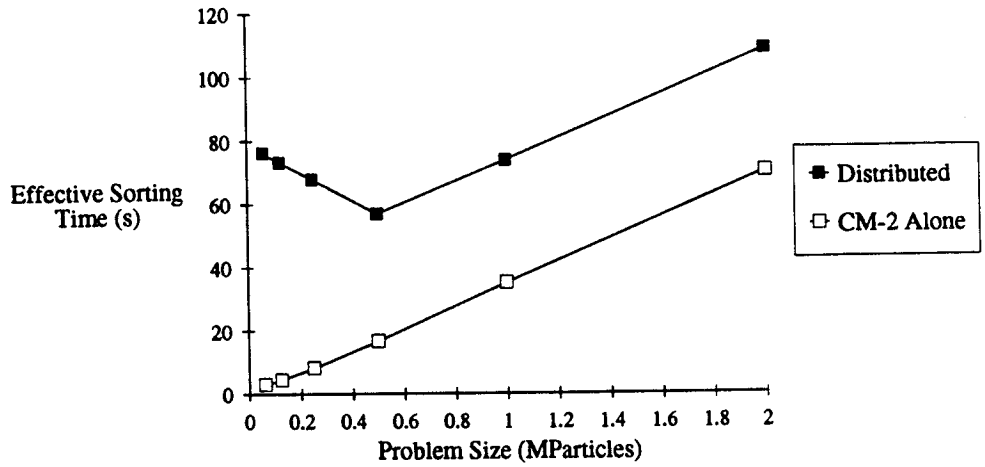

Figure 6: Effective Sorting Time, One Sequencer

single sequencer times, the distributed times do not appear to change much from the single sequencer times. There are two reasons for this. First, the Y-MP time does not change since the same quantity of data is worked on regardless of the number of sequencers used. The Y-MP bucket sort is still faster than the stand-alone CM-2 sort. Second, for problem sizes smaller than two million particles, the network transfer time is not affected by the addition of a sequencer and hence, an I/O bus.

As Figure 8 shows, the data conversion time for the distributed code is not affected much by the number of sequencers. It only begins to be linear for problem sizes larger than half a million particles. A detailed inspection of the individual Y-MP times shows that they vary linearly with problem size, and are independent of the number of sequencers. Thus, any nonlinearity is due to the CM-2 transpose operation. The fact that the single and double sequencer times are so close for small problem sizes implies that there is a large overhead on the CM-2 side.

Data conversion is a significant overhead factor. Comparing Figures 8, 4 and 5, data conversion takes between a quarter and a third of the total time for large enough problem sizes. For smaller problem sizes, the network overhead dominates the data conversion time. We expect that data conversion overhead will be reduced in the future. We have developed an optimized version of the code in which the Y-MP data conversion is vectorized. Al-

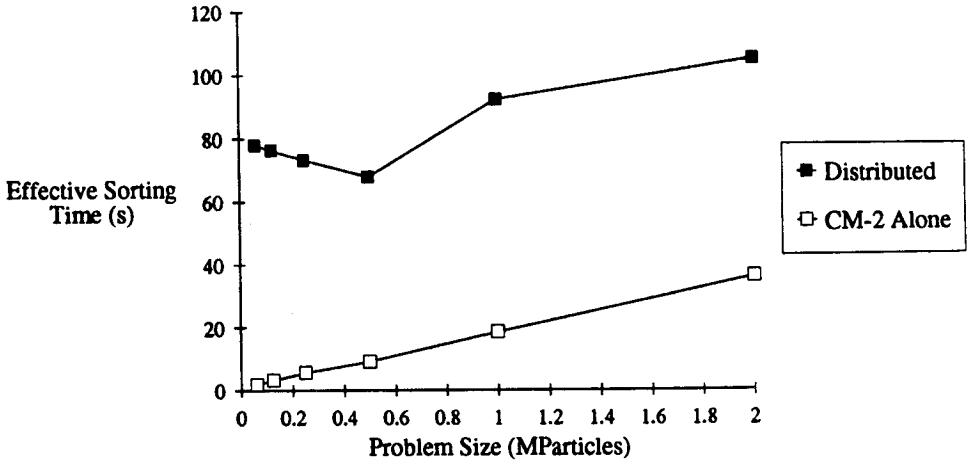

Figure 7: Effective Sorting Time, Two Sequencers

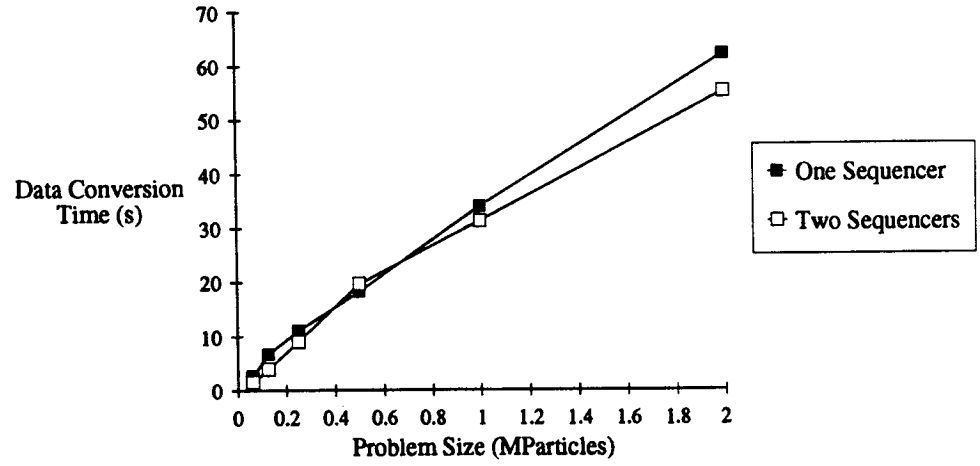

Figure 8: Data Conversion Time

though we have not had a chance to test it extensively, preliminary runs show speedups of 80 in the Y-MP data conversion times. In addition, TMC has announced that the new version of their operating system contains an optimized transpose procedure with twice the speed of the current version.

As shown in Figure 9, the network transfer times for both single and double sequencer runs show a decreasing trend with larger problem sizes. This is very counter-intuitive, given that the amount of data transferred grows linearly with problem size. One possible explanation is that computation is overlapped with I/O by the optimizer. Since our timers are measuring processor time, any overlapped I/O time is hidden. As the problem size gets bigger, the amount of I/O that can be overlapped with processor time goes up.

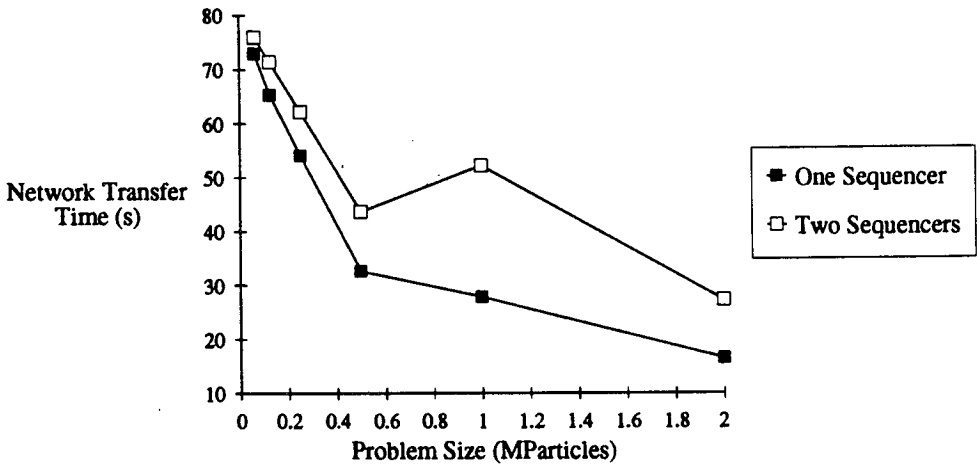

Figure 9: Network Transfer Time

It is curious that the times for runs with two sequencers are longer than those with one. However, this may also be due to the amount of overlap of I/O and processor time. With two sequencers, each sequencer is performing half as much I/O as in the one sequencer case for a given problem size. Thus the one sequencer case has more I/O that it can overlap with processor time. Further experimentation is required to find out if these conjectures are true. Note that at the largest problem size for one sequencer, a total of 160 MBytes are transferred in about 16 seconds, yielding an effective transfer rate of 10 MBytes/s, a very reasonable bandwidth.

We have noticed similar transfer rate behavior using the DataVault instead of the CM-HIPPI. The DataVault is the disk system connected to the CM-2 through the I/O busses. We ran experiments on one sequencer which used identical transfers to those in the particle simulation. For small problem sizes, the time dropped as problem size was increased. For larger problem sizes, transfer time increased with problem size. Although this is not exactly the same behavior observed in the network transfers (Figure 9), both cases have different transfer time variation for small and large problem sizes, with a transition at a ratio of virtual to physical processors equal to 64. This implies that different techniques are used by the CM-2 I/O system for small and large transfers respectively.

We ran another set of tests in which 512 bytes were transferred from each virtual processor to the DataVault. Increasing the number of sequencers caused an increase in transfer time, as in the network transfer measurements of Figure 9. However, the DataVault may only be accessed by a single I/O bus regardless of the number of sequencers used. Increasing the number of sequencers also raises the number of processors. Thus, there is more contention for the single bus and the transfer rate drops. This implies that doubling the number of sequencers in the particle simulation may not have allowed access to two I/O busses as expected.

5 Summary and Future Research

We have written the first application at NASA Ames to distribute a computation between a CM-2 and a Y-MP over HIPPI and UltraNet. The results of the particle simulation tests are quite encouraging. Although we were not able to achieve the speed of the original stand-alone CM-2 implementation, the distributed computation was within a factor of two to four for problem sizes larger than half a million particles. In addition, the results enabled us to understand the effects of various overheads. Network overheads dominate for small problems, as expected. It seems that a distributed implementation requires large transfer sizes to be feasible. Data conversion is expensive, comprising up to one third of the total time. As described earlier, we expect to reduce this overhead in future versions.

Our particle simulation did not match the performance improvement attained by the PSC application. However, they were able to run large prob-

lems comprised of 64 million elements. Thus the overheads incurred by distributing the application were not a significant factor. We were unable to run problems larger than about 2 million particles due to several problems discussed in Section 3. Ideally, we would like to scale our problem to the maximum memory capacity of the CM-2 and Y-MP. For the systems at NAS, this should permit simulations of up to about 30 million particles. Problems of this size should have greatly improved performance over the simulations we have been able to run to date.

Running large problems will not be possible until all four CM-2 sequencers, and hence, all 32K processors, can be used with the HIPPI link. Since our application is interactive, it is appropriate to run the Y-MP code out of the interactive NQS queue, instead of the batch queues. In addition, it is difficult to synchronize the sockets using the batch queues. However, this limits us to problems of size equal to about 4 million particles. It might be useful for future NQS versions to include a second interactive queue with larger processor time and memory allocation in addition to the current interactive queue. Finally, the intermittent hanging of the network must be repaired before we can run our application more dependably.

Our main goal for future research is to use our optimized code to run larger problems. In addition, we plan to improve the application's ease of use by employing a daemon-based system for the sockets. Finally, we plan to experiment with other hardware platforms, and other applications. For example, some large heterogeneous applications might require more than two hardware platforms in order to run effectively. This research will result in the development of high performance HDS systems which greatly extend the capabilities of current supercomputing platforms.

6 Acknowledgements

We would like to thank Ken Stevens for his helpful input and for providing access to NAS facilities to one of us (JB) during a recent internship at NASA Ames. We would also like to thank Russell Carter for his help with various aspects of this project, John Krystynak for running the DataVault tests, and Chris Kuszmaul for his work on connecting the CM-2 and Y-MP with sockets.

7 References

[**Dagum91**] L. Dagum, Three Dimensional Direct Particle Simulation On the Connection Machine, *AIAA 26th Thermophysics Conference*, paper 91-1365, 1991.

[**Schneider91**] M. Schneider, Pittsburgh's Not-So-Odd Couple, *Supercomputing Review*, Vol. 4, No. 8, pp. 36-38, 1991.